《数据结构与算法》知识点整理

来自 Xzonn 的小站

更新于 2020-01-03 23:02 · 渲染于 2021-01-11 15:29

目 录

1 概述

1.1 数据结构

- 数据结构：抽象数据类型的物理实现。以主机的运行时间和内存的存储空间来权衡。
- 数据结构三要素：
- 逻辑结构：基本元素和元素之间的相互关系。
- 存储结构：具体表现方式，包括基本元素的表示和关系的表示。
- 操作：各种行为在存储结构上的具体实现算法。
- 数据结构的分类：
- 按逻辑结构分类：逻辑结构可用二元组 $B = \langle K, R \rangle$ 来表示，K 是结点的有穷集合，R 是 K 上的一个关系。K 上的二元组是 K 中元素的有序对，记为 $\langle k, k' \rangle$。k 是 k' 的前驱，k' 是 k 的后继。根据 R 的特点分为线性结构（每个结点最多一个前驱和一个后继）；树形结构（每个结点最多一个前驱，可有多个后继）；复杂结构（前驱、后继结点个数不限）。
- 按存储结构分类：顺序表示；链接表示；散列表示；索引表示。

1.2 算法

- 算法：由有穷规则构成的为解决某一类问题的运算序列（方法或过程）。可以由若干输入，通常有若干个输出。
- 算法的性质：
- 有穷性：一个算法必须在执行了有穷步后结束。
- 确定性：算法的每一步需要执行的动作必须严格清楚地给出规定。
- 可行性：算法中的每个动作原则上都能由机器或人准确完成。
- 算法的正确性：如果一个算法以一组满足初始条件的输入为开始，那么算法的执行一定会终止并得到满足要求的结果。
- 算法的设计方法：
- 贪心法：将整个问题分成若干阶段，每一个阶段都选择局部最优方案。

- 分治法：将规模较大问题分成几个较小问题，求解子问题再合并子问题的解，如二分法。
- 回溯法：采用一步一步向前试探的方法，当某一步有多种选择时先任选一种继续向前，无法前进时后退回上一层，即深度优先策略，如迷宫问题。
- 动态规划法：与分治法类似，但分解的子问题较多且子问题相互包含，需要保存计算的中间结果，通常自底向上进行。
- 分枝界限法：与回溯法类似，但采取广度优先策略，利用最优解属性的上下界控制分枝。

2 算法分析

- 大 O 表示法：若某个算法的代价为 $T(n) = O(f(n))$，则存在常数 $c > 0$，$N > 0$，当 $n > N$ 时该算法的代价 $T(n) \le c \cdot f(n)$。一般使用上确界。
- 其它表示法：
- $T(n) = \Omega(f(n))$ 表示法：$T(n) \ge c \cdot f(n)$。
- $T(n) = \Theta(f(n))$ 表示法：$T(n) = O(f(n))$ 且 $T(n) = \Omega(f(n))$。

3 基本数据结构

3.1 线性表

- 线性表：简称表，可用二元组 $L = \langle K, R \rangle$ 表示，$K = \{k_0, k_1, \cdots, k_{n-1}\}$，$R = \{\langle k_i, k_{i+1} \rangle \mid 0 \le i \le n-2\}$。结点之间满足线性关系，第一个元素仅有一个后继，最后一个元素仅有一个前驱，其他元素仅有一个前驱和一个后继。
- 顺序表示：假设每个元素沿用 c 个存储单元，则 $\mathrm{loc}(k_i) = \mathrm{loc}(k_0) + i \times c$。只要确定了首地址，线性表中的元素可以随机存储。
- 时间复杂度：插入删除 $O(n)$，无序查找 $O(n)$，有序查找 $O(\log n)$，取值 $O(1)$。
- 链接表示：每个结点包括数据域（存放元素信息）和指针域（指向后继元素）。每个结点只有一个指针域的链表为单链表。有时为了处理方便可以在单链表的第一个结点前加一个头结点。
- 时间复杂度：插入删除 $O(1)$，查找取值 $O(n)$。
- 循环链表：最后一个结点的指针指向第一个结点。从任一结点出发都能访问所有结点。
- 双链表：每个结点保存前驱和后继。克服单链表单向性的缺点。此外还有循环双链表。
- 有序表：数据项依照其可比性质（如整数大小）来决定在列表中的位置。对于有序表可以利用结点有序排列的特点节省查找时间，但添加时必须比较数据项选择合适位置插入。
- 顺序表与链表比较：
- 顺序表示优点：随机存取任一元素；缺点：插入删除效率低，估计最大空间困难。
- 单链表存储密度比顺序表低；插入删除效率高。

3.2 栈和队列

- 栈：所有的插入和删除都限制在表的同一端进行。允许操作的一端为栈顶，另一端为栈底。无元素的栈称为空栈。特点：后进先出。
- 队列：只允许在表的一端进行插入，在另一端删除。允许删除的一端为表头，允许插入的一端为队尾。无元素的队列称为空队。特点：先进先出。
- 队列的实现：环形队列，维护头结点和尾结点。

- 双端队列：数据项可以从两端分别插入和删除。集成了栈和队列的能力。

4 递归

- 递归：函数自己调用自己的做法。三定律：
- 递归算法必须有一个基本结束条件（最小规模问题的直接解决）。
- 递归算法必须能改变状态向基本结束条件演进（减小问题规模）。
- 递归算法必须调用自身（解决减小了规模的相同问题）。
- 递归调用的实现：栈。
- 动态规划：保存计算的中间结果。

5 排序与搜索

- 散列法：选择一个从关键码到地址的映射函数 h（散列函数），对于每个关键码为 key 的元素，计算 $h(\text{key})$（散列地址），期望把对应的元素存放到该地址。
- 碰撞：不相等的两关键码经散列函数计算得到相同散列地址。
- 完美散列函数：给定一组关键码，散列函数能把每个关键码映射到不同的地址。
- 负载因子 α：$\alpha = \dfrac{\text{字典中节点数目}}{\text{基本区域能容纳的结点数}}$。
- 散列函数设计：
- 三个特性：冲突最少、计算难度低、充分分散数据项。
- 求余数：关键码除以散列表大小，将余数作为地址。
- 折叠法：将数据项按照位数分为若干段，再将几段数字相加，最后对散列表大小求余，得到散列值。
- 平方取中法：首先将数据项做平方运算，然后取平方数的中间两位，再对散列表的大小求余。
- 冲突解决方案：
- 开地址法（开放寻址法）：在存储区域内形成探查序列，沿此序列逐个查找，知道找到要查找的元素或碰到未被占用的地址。线性探查法：即从冲突位置向后逐个扫描。也可每次增加 skip 个地址，但需要保证 skip 与散列表大小互质。
- 拉链法：在每个地址中开辟一个链表，先由 $h(\text{key})$ 确定数据项在哪一条链表中，再在链表中进行插入、删除、检索等操作。
- 排序算法：
- 冒泡排序：对无序表进行多次比较交换，每次两两相邻比较，并将逆序数据项互换位置。时间开销为 $O(n^2)$，空间开销为 $O(1)$。
- 选择排序：每次比较记录最大项位置，最后与本次比较最后一项交换顺序。时间开销为 $O(n^2)$，空间开销为 $O(1)$。不稳定。
- 插入排序：维持一个已经排好序的子列表，每次将下一个数据项插入已排序列表中。时间开销为 $O(n^2)$，空间开销为 $O(1)$。
- 希尔排序：将整个列表按照 d_1 间隔分割为几个小列表，在小列表内排序，再取间隔为 d_2（$d_2 < d_1$）分割列表，直到 $d_n = 1$。时间开销为平均 $O(n \log^2 n)$，最坏 $O(n^2)$，空间开销为 $O(1)$。
- 归并排序：利用递归算法，将列表分为两半，分别调用归并排序算法，直到子列表仅有一个项；然后按照大小顺序合并两个子列表。时间开销为 $O(n \log n)$，空间开销为 $O(n)$。

- 快速排序：选取 “中值” 将列表分为两半，左边项均小于中值，右边项均大于中值，然后在左右列表递归调用快速排序算法。时间开销为平均 $O(n\log n)$，最坏 $O(n^2)$，空间开销为 $O(\log n)$。

6 树及其算法

- 树：由一个根结点和几棵互不相交的子树组成。二叉树：有两棵子树，分别为左子树和右子树。概念：
- 父结点、左（右）子结点、边：若 x 是二叉树的根结点，y 是 x 左（右）子树的根结点，则 x 是 y 的父结点，y 是 x 的左（右）子结点，有序对 $\langle x, y\rangle$ 称为从 x 到 y 的边。
- 兄弟、祖先、子孙：具有同一父节点的结点彼此为兄弟。若结点 y 在以结点 x 为根的左（右）子树中且 $y\neq x$，则 x 是 y 的祖先，y 是 x 的子孙。
- 树叶、分支结点：左右子树均为空二叉树的结点称为树叶，否则称为分支结点。
- 路径、路径长度：若 x 是 y 的祖先，存在节点序列 $x_0, x_1, \cdots, x_n$ 满足 $x_0 = x$，$x_n = y$，则称该序列为从 x 到 y 的一条路径，n 称为路径长度。
- 层数、高度：规定根的层数为 0，其余结点的层数是父节点层数加 1。二叉树中结点的最大层数称为二叉树的高度（深度）。结
- 度数：点的非空子树的个数叫做结点的度数。二叉树每个结点度数最大为 2。
- 特殊的二叉树：
- 满二叉树：若一棵二叉树的任何结点或者是树叶或是两棵非空子树，则称为满二叉树。
- 完全二叉树：若一棵二叉树中只有最下面两层结点度数小于 2，其余各层结点度数都等于 2，且最下面一层的结点都集中在最左边，则称为完全二叉树。
- 二叉树的实现：嵌套列表法、结点链接法。
- 树的周游（遍历）：深度优先周游、广度优先周游。
- 深度优先周游：按照访问根节点的顺序，分为先根次序（前序遍历）、后根次序（后序遍历）、中根次序（中序遍历）。
- 广度优先周游：从 0 到 h 逐层从左往右访问每个结点。
- 实现：递归算法。
- 堆：对 n 个元素的序列，若满足 $\begin{cases} k_i \leq k_{2i+1}, \\ k_i \leq k_{2i+2}, \end{cases}$ 则称次序列为最小堆。可以用完全二叉树实现堆。
- 优先队列：遵循 “最小元素先出” 的规则。通过堆实现优先队列：
- 加入优先队列：先把新元素放在最后位置，再通过与父节点比较交换结点顺序，直到堆序性满足。
- 删除最小元素：先删除根节点，然后将最后一个结点放入根节点位置，再与子节点比较交换节点顺序，直到堆序性满足。
- 二叉排序树（二叉搜索树）：每个父结点的左子树结点值都比父节点小，右子树结点值都比父节点大。
- 检索：类似二分法，比较要查找的结点与左右子节点的大小关系，然后在左右子树中搜索。
- 插入：若根节点为空则插入根节点，否则若插入值等于根节点值则已存在，若插入值小于根节点值则插入左子树，否则插入右子树。
- 删除：找到被删除的结点，若其没有左子树，则将其右子树代替被删除结点。否则找到左子树中最右下的结点（在左子树中值最大），让被删除结点的右子树称为该结点的右子树，再让被删除节点的左子树代替被删除结点。
- 最佳二叉排序树：在检索过程中平均比较次数 $E(n)$ 最小的二叉排序树。

- 构造（各结点等概率）：现将所有元素排序，然后对每个元素的值按二分法检索，将检索中遇到的还未在二叉排序树中的元素插入二叉排序树中。
- 平衡二叉排序树（AVL 树）：每个结点左右子树高度之差的绝对值不超过 1 的二叉排序树。
- 平衡因子：结点右子树高度与左子树高度之差。
- 插入：若新结点插入不影响父结点为根的树的高度，则不破坏平衡；否则应调整。主要手段：将不平衡的子树进行旋转。
- 最小不平衡子树：离插入结点最近且根节点平衡因子绝对值大于 1 的树。
- 调整平衡：左重则先检查左子节点是否右重，若右重则先左旋转，然后原结点右旋转；右重相反。

7 图及其算法

- 图：由结点和边组成。有向图：每条边有方向。有向边表示为 $\langle v_i, v_j\rangle$。无向图：每条边无方向。无向边表示为 (v_i, v_j)。概念：
- 关联、邻接：若 $\langle v_i, v_j\rangle$ 是有向边，则顶点 v_i 邻接到 v_j 或 v_j 邻接于 v_i，边 $\langle v_i, v_j\rangle$ 与顶点 v_i，v_j 相关联。若 (v_i, v_j) 是无向边，则 v_i 和 v_j 是相邻结点。
- 度、入度、出度：与顶点 v 相关联的边数称为度，记为 $D(v)$。有向图以 v 为终点的边数称为入度 $ID(v)$，为始点的边数称为出度 $OD(v)$。
- 子图：设图 $G = (V, E)$ 和图 $G' = (V', E')$，若 V' 是 V 的子集，E' 是 E 的子集，则 G' 是 G 的子集。
- 根与有根图：有向图中若存在顶点 v，从该顶点有路径可到达图中其他所有顶点，则称此图为有根图，v 是图的根。
- 连通图、连通分量：无向图 $G = (V, E)$ 中若从 v_i 到 v_j 之间有一条路径，则 v_i 和 v_j 是连通的。若 $V(G)$ 中任意两个不同的顶点都是连通的，则 G 是连通图。无向图 G 中的最大连通子图称为 G 的连通分量。
- 强连通图、强连通分量：有向图 $G = (V, E)$ 中若 $V(G)$ 中任意两个不同的顶点都是连通的，则 G 是强连通图。有向图 G 中的最大连通子图称为 G 的强连通分量。
- 实现：
- 邻接矩阵：用二维矩阵，每行每列代表图中的顶点，若两个顶点之间连通则再相应行列值的矩阵分量中加以体现。
- 邻接列表：维护一个包含所有顶点的主列表，主列表中的每个顶点关联一个与自身连通的所有顶点的列表。
- 图的周游：深度优先周游（DFS）、广度优先周游（BFS）。
- 强连通分量算法：先对图 G 调用深度优先周游算法，为每个顶点计算结束时间，然后对 G 进行转置，得到 G^T，再对 G^T 调用深度优先周游算法（以结束时间倒序搜索），最后深度优先森林中的每一棵树就是一个强连通分支。
- Dijkstra 算法：求顶点 v_0 到 v_n 的最短路径。维护两个集合 U 和 S，其中 U 为已求出从 v_0 到它最短路径的顶点，$S = V - U$ 存放未确定最短路径的顶点。初始时 U 中只有 v_0，其路径长度为 0，S 中为其它所有顶点，且与 v_0 直接相连的顶点路径长度已知，不直接相连的顶点路径长度为 $+\infty$。每次从 S 中选取路径最短的顶点加入 U 中，并求出 v_0 通过 U 中顶点到达 S 中顶点的最短路径，重复操作直到 v_n 在 U 中。用优先队列实现可以使时间开销为 $O((|V| + |E|) \log |V|)$。
- 最小生成树：对于连通的无向图或强连通的有向图，从任一顶点周游可以访问图中所有顶点，周游时形成的边称为图的一棵生成树。将生成树各边的权值加起来称为生成树的权，把权值最小的生成树称为最小生成树（MST）。

- Prim 算法：设 $T = (U, TE)$ 是最小生成树，初始状态 T 为空树，先从 V 中任取一顶点放入 U 中，然后在所有一个顶点在 U 中、另一个顶点在 $V - U$ 中的边中选取权最小的边放入 TE，重复上述过程直到 $U = V$。